|  |  |
| --- | --- |
| ***Micro-instruction*** | **Description** |
| ***stctrljpc*** | Stores the address (JPC) of next byte-code pushed onto top of stack (TOS) A into the ***ctrljpcreg***register. This is saved when we are executing the CRCF instruction from the control memory and want to shift the execution control to DP.  A 🡪 ctrljpcreg |
| ***stmainptr*** | Stores the address of the main method structure present at TOS in the ***mainprtreg***register. The method structures contain the information such as start address of the byte code, its length and constant pool address.  A 🡪 mainptrreg |
| ***stmainjpc*** | Stores the address of next byte-code loaded onto TOS A, into the ***mainjpcreg***register. This is saved when we shift the execution DP to CRCF.  A 🡪 main jpcreg |
| ***ldt*** | Pushes the content of a temporary register **T** onto **TOS A**.  T 🡪 A |
| ***wrctrl*** | It generates ***wr\_en*** signal for control memory for storing data provided address. The address and data are provided as the top two elements (A & B) of stack.  CM[A] 🡨B |
| ***rdrf*** | Loads the contents of register file into ***regx & regz***.  Rf[sel\_x] 🡪 regX, Rf[sel\_z] 🡪 regZ |
| ***wrrf*** | It generates the ***wr\_en*** signal for the writing the data to register file. The contents to be written are provided on the TOS and address of registers is part of instruction (16 bit).  Rf[sel\_z] 🡨 A |
| ***ldctrljpct*** | Loads the content of ***ctrljpcreg*** into the temporary register.  Cltrjpcreg 🡪 T |
| ***toggle*** | Sets and clears the ***ctrl*** flag. When set, CRCF instructions are executed else DP instructions are executed.  Ctrl 🡨 not ctrl |
| ***ldmainjpct*** | Loads the content of ***mainjpcreg*** into the temporary register ***T***.  mainjpcreg 🡪 T |
| ***ldctrlopdt*** | Loads the control operand into the temporary register ***T***.  Cltropdreg 🡪 T |
| ***ldmainptrt*** | Loads the content of ***mainjptrreg*** into the temporary register ***T***.  ldmainptrreg 🡪 T |
| ***ldrzt*** | Loads the content of register Z into the temporary register ***T***.  regZ 🡪 T |
| ***ldrxt*** | Loads the content of register X into the temporary register ***T***.  regX 🡪 T |
| ***wrdm*** | Write signal for data memory.  Dm[A] 🡨 B |
| ***lddmt*** | Loads the data read from control data structure memory into the temporary register ***T***.  Dm[A] 🡪 T |
| ***ldmaxt*** | Loads the result of max unit into the temporary register ***T***. The max unit finds the maximum nibble value.  Maxreg 🡪 T |
| ***seot*** | Sets the ***EOT*** register.  EOT 🡨 1 |
| ***ceot*** | Clears the ***EOT*** register  EOT 🡨 0 |
| ***ldert*** | Loads the content of ***ER*** register into temporary register.  ERreg 🡪 T |
| ***cer*** | Clears the ***ER*** register.  ERreg 🡨 0 |
| ***ssvop*** | Loads the ***SVOP*** register with the contents of ***regX***  SVOP 🡨 regX |
| ***sop*** | Loads the ***SOP*** register with the contents of ***regx***  SOP 🡨 regX |

Continued on next page

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Byte-code** | **Microinstruction** | **Register Transfer** |
| **1** | **jopsys\_exectrl**  **DPCF**  This byte-code shifts the execution control to CRCF. It stores the execution address of main in a register for a future use. The CRCF address is loaded into program counter and control falg is toggled. | ldjpc  stmainjpc  ldctrlpct  ldt  stjpc  nop  toggle  nop  nop  nop  nop nxt | Jpc 🡪 A  A🡪mainjpcreg ( we do not need it now)  Ctrlpcreg🡪 T  T🡪 A  A🡪 jpc  Ctrlflag 🡪 not ctrlflag |
| **2** | **jopsys\_rtc(result)**  **(DPCF)**  This bytecode returns directly to CRCF from DPCF without reloading methodcache and also writes the result of computation back into memory. The results is provided as argument and is available in A whereas register Rx contains method\_id and datalock position. The data lock position is extracted and result is written to memory location pointed to by datalock | nop  ldrxt  ldt  ldi 255  and  wrdm  nop  ldctrlpct  ldt  stjpc  nop  pop  pop  nop  toggle  nop  nop  nop nxt | Rx 🡪 T  T 🡪 A  A 🡪 B, 255 🡪 A  A and B 🡪 A (A= datalock, B=result)  B 🡪 dm[A]  Ctrlpcreg🡪 T  T🡪 A  A🡪 jpc  Remove A and B ( datalock and result)  Ctrlflag 🡪 not ctrlflag |
| **3** | **jopsys\_stmainp(mainp)**  **DPCF**  main pointer is provided in A from where it is stored in mainptrreg | stmainptr  nop  nop nxt | A 🡪mainptrreg |
| **4** | **jopsys\_initctrl(instr, address)**  **DPCF**  This bytecode is used to initialized the CRCF memory during the execution of the main. The instruction and address are pushed on TOS and TOS-1. | wrctrl  nop  pop  pop  nop nxt | B 🡪 cm[A] (B= instruction, A= address)  Remove the address and data |
| **5** | **jopsys\_nopinstr:**  **CRCF** | nop  nop nxt |  |
| **6** | **jopsys\_datacall:**  **CRCF**  This CRCF instruction is used to directly invoke the desired data computation directly from within the CRCF instead of going to main and then invoking the desired method conventionally. The address of the main method data structure is pushed on the stack which is used to extract calculate the address of the data structure of the desired method as all theses address are relative. This put in a restriction on the declaration of the data computation methods should be ordered which makes it possible to calculate the address using the method\_id provided in a register of register file concatenated with datalock position. The contents of register are pushed on the TOS and method\_id is extracted which is duplicated and subtracted from mainpreg-2 to get the address of data method containing the required computation. The CRCF PC is saved. The data structure of the method is read which is two words in length. The second word is read first which contains the number of arguments, local variable count and constant pool address. The first word of method structure contains the information about the method code such as the start address and code length. The information is passed on to the memory subsystem for loading the desired method in the method cache. The start address of the method is cache is pushed onto TOS from where it is popped into JPC and ctrl flag is toggled resulting in execution of DPCF method. | rdrf  ldmainptrt  ldt  ldrxt  ldt  ldi 8  ushr  dup  add  sub  ldjpc  nop  stctrlpc  dup  nop  ldi 1  add  stmrac  stm mp  wait  wait  ldmrd  nop  ldm mp  stmrac  wait  wait  ldmrd  stbcrd  dup  ldi 31  and  stm args  ldi 5  ushr  dup  ldi 31  and  stm varcnt  ldi 5  ushr  stm cp  ldsp  ldi 1  add  dup  ldm args  sub  stm old\_sp  ldm old\_sp  ldi 1  add  stvp  add  nop  stsp  pop  pop  nop  ldbcstart  stjpc  nop  nop  toggle  nop  nop  nop nxt | Mainptrreg 🡪 T  T 🡪 A , A= mainptrreg  Rx 🡪 T, B=mainptrreg  T 🡪 A (Rx), B=mainptrreg  8 🡪 A, B=Rx  A 🡪 A>>8 A= method\_id  A 🡪 A, A🡪 B  A+B 🡪 A(method\_id+method\_id) , B=mainptr  A= B-A (mainptr – 2\* method\_id)  JPC 🡪 A  A 🡪 ctrlpcreg  A🡪 A, A🡪 B, A=B= method address  1 🡪 A  A 🡪 A+1  Cpword 🡪 A  Method address=mp 🡪 A  Methodinfoword 🡪 A  A 🡪 memory subsystem ( start loading into cache)  Extract constan pool info  Method start address 🡪 A  A 🡪 JPC  Ctrl flag 🡪 ctrl flag |
| **7** | **jopsys\_strimm:**  **CRCF**  This instruction stores the immediate operand into memory location pointed to by the contentsof register Rz. The operand to be written and address provided by Rz are pushed on the stack in the same order. After write operations is completed, the TOS and TOS-1 are flushed. | nop opd  rdrf  ldctrlopdt  ldt  ldrzt  ldt  wrdm  nop  pop  pop nxt | Opd 🡪 T  T🡪 A  Rz 🡪 T  T🡪 A (Rz=memory address), B(data)  B 🡪 dm[A] |
| **8** | **jopsys\_strind:**  **CRCF**  This instruction stores the contents of register Rx into memory location pointed to by the contents of register Rz. The contents of register Rx which are to be written are pushed on the stack together with address provided by Rz. After write operation is completed, the TOS and TOS-1 are flushed. | rdrf  ldrxt  ldt  ldrzt  ldt  wrdm  nop  pop  pop nxt | Rx 🡪 T  T🡪 A(=data)  Rz 🡪 T  T🡪 A (Rz=memory address), B(data)  B 🡪 dm[A] |
| **9** | **jopsys\_strdir:**  **CRCF**  This instruction stores the contents of register Rx into memory location pointed to by the immediate operand. The contents of register Rx which are to be written are pushed on the stack together with address. After write operation is completed, the TOS and TOS-1 are flushed. | nop opd  rdrf  ldrxt  ldt  ldctrlopdt  ldt  wrdm  nop  pop  pop nxt | Rx 🡪 T  T🡪 A(=data)  opd 🡪 T  T🡪 A (opd=memory address), B(data)  B 🡪 dm[A] |
| **10** | **jopsys\_ldrimm:**  **CRCF**  The immediate operand is loaded into the register Rz. The CRCF instruction is 16-bit long. The msb byte represents opcode and lsb byte contains the address for Rz and Rx respectively. | nop opd  nop  wrimm  nop  nop nxt | Ctrlopd 🡪 RF[z] |
| **11** | **jopsys\_ldrind:**  **CRCF**  This instruction loads the content of memory location pointed to by the register Rx into the register Rz. The contents of Rx are pushed on stack which is address to memory containing the data structure of CRCF. The memory contents are read into T and pushed on to stack for writing to resgiter. The TOS and TOS-1 are flushed afterwards. | rdrf  ldrxt  ldt  nop  lddmt  ldt  nop  wrrf  nop  nop  pop  pop nxt | Rx 🡪 T  T🡪 A(=address)  dm[A] 🡪 T  T🡪 A  A 🡪 RF[z] |
| **12** | **jopsys\_ldrdir**  **CRCF**  This instruction loads the content of memory location pointed to by the immediate operand into the register Rz. The contents of operand are read from program memory into register T and are pushed on stack which is address to memory containing the data structure of CRCF. The memory contents are read linto T and pushed on to stack for writing to resgiter. The TOS and TOS-1 are flushed afterwards. | nop opd  nop  ldctrlopdt  nop  ldt  nop  lddmt  ldt  nop  wrrf  nop  pop  pop nxt | ctrlopd🡪 T  T🡪 A(=address)  dm[A] 🡪 T  T🡪 A  A 🡪 RF[z] |
| **13** | **jopsys\_andimm**  **CRCF**  It performs AND operation between the immediate value and content of register Rx and stored the result into the register Rz. The content register Rx and operand values are pushe onto stack and AND operand is performed. The result available on TOS is written back to register Rz pointed to by instruction. The TOS is removed once we have written to register file. | nop opd  rdrf  nop  ldrxt  ldt  ldctrlopdt  ldt  and  wrrf  nop  pop nxt | Rx 🡪 T  T🡪 A(=Rx)  ctrlopd🡪 T  T🡪 A (=ctrlopd), B= Rx  A 🡪 A and B (= Rx+ctrlopd)  A 🡪 RF[z] |
| **14** | **jopsys\_orimm**  **CRCF**  It performs OR operation between the immediate value and content of register Rx and stored the result into the register Rz. The content register Rx and operand values are pushe onto stack and OR operand is performed. The result available on TOS is written back to register Rz pointed to by instruction. The TOS is removed once we have written to register file. | nop opd  rdrf  nop  ldrxt  ldt  ldctrlopdt  ldt  or  wrrf  nop  pop nxt | Rx 🡪 T  T🡪 A(=Rx)  ctrlopd🡪 T  T🡪 A (=ctrlopd), B= Rx  A 🡪 A or B (= Rx+ctrlopd)  A 🡪 RF[z] |
| **15** | **jopsys\_addimm**  **CRCF**  It performs ADD operation between the immediate value and content of register Rx and stored the result into the register Rz. The content register Rx and operand values are pushe onto stack and ADD operand is performed. The result available on TOS is written back to register Rz pointed to by instruction. The TOS is removed once we have written to register file. | nop opd  rdrf  nop  ldrxt  ldt  ldctrlopdt  ldt  add  wrrf  nop  pop nxt | Rx 🡪 T  T🡪 A(=Rx)  ctrlopd🡪 T  T🡪 A (=ctrlopd), B= Rx  A 🡪 A+B (= Rx+ctrlopd)  A 🡪 RF[z] |
| **16** | **jopsys\_subvimm**  **CRCF**  It performs SUB operation between the immediate value and content of register Rx and stored the result into the register Rz. The content register Rx and operand values are pushe onto stack and SUB operand is performed. It affects the Zero flag.The result available on TOS is written back to register Rz pointed to by instruction. The TOS is removed once we have written to register file. | nop opd  rdrf  nop  ldrxt  ldt  ldctrlopdt  ldt  sub  nop  stzf  wrrf  nop  pop nxt | Rx 🡪 T  T🡪 A(=Rx)  ctrlopd🡪 T  T🡪 A (=ctrlopd), B= Rx  A 🡪 A-B (= Rx+ctrlopd) (set ZF if A=0)  A 🡪 RF[z] |
| **17** | **jopsys\_andind**  **CRCF**  It performs AND operation between the content of register Rx and RZ and stores the result back into the register Rz. The content register Rx and RZ are pushed onto stack and SUB operand is performed. The result available on TOS is written back to register Rz pointed to by instruction. The TOS is removed once we have written to register file. | rdrf  nop  ldrxt  ldt  ldrzt  ldt  and  wrrf  nop  pop nxt | Rx 🡪 T  T🡪 A(=Rx)  Rz🡪 T  T🡪 A (=Rz), B= Rx  A 🡪 A and B (= Rx and rz)  A 🡪 RF[z] |
| **18** | **jopsys\_orind**  **CRCF**  It performs OR operation between the content of register Rx and RZ and stores the result back into the register Rz. The content register Rx and RZ are pushed onto stack and OR operand is performed. The result available on TOS is written back to register Rz pointed to by instruction. The TOS is removed once we have written to register file. | rdrf  nop  ldrxt  ldt  ldrzt  ldt  or  wrrf  nop  pop nxt | Rx 🡪 T  T🡪 A(=Rx)  Rz🡪 T  T🡪 A (=Rz), B= Rx  A 🡪 A or B (= Rx or rz)  A 🡪 RF[z] |
| **19** | **jopsys\_addind**  **CRCF**  It performs ADD operation between the content of register Rx and RZ and stores the result back into the register Rz. The content register Rx and RZ are pushed onto stack and ADD operand is performed. The result available on TOS is written back to register Rz pointed to by instruction. The TOS is removed once we have written to register file. | rdrf  nop  ldrxt  ldt  ldrzt  ldt  add  wrrf  nop  pop nxt | Rx 🡪 T  T🡪 A(=Rx)  Rz🡪 T  T🡪 A (=Rz), B= Rx  A 🡪 A+B (= Rx+rz)  A 🡪 RF[z] |
| **20** | **jopsys\_jmpimm**  **CRCF**  This instruction result in an unconditional jump to the location provided as immediate value. The operand is read into T and pushed on TOS. It is stored into JPC and the next instruction is fetched from the address pointed to by the immediate. | nop opd  nop  ldctrlopdt  nop  jbr  nop  nop  nop nxt | Ctrlopd 🡪 T  T 🡪 JPC |
| **21** | **jopsys\_jmpind**  **CRCF**  This instruction result in an unconditional jump to the location indicated by the contents of register Rx. The contents of register Rx are read into T and pushed on TOS. It is stored into JPC and the next instruction is fetched from the address pointed to by the immediate. | rdrf  nop  ldrxt  nop  jbr  nop  nop  nop nxt | Rx🡪 T  T 🡪 JPC |
| **22** | **jopsys\_noop**  **CRCF** | nop  nop nxt |  |
| **23** | **jopsys\_present**  **CRCF**  This instruction performs a conditional jump. The contents of the register Rz are checked, if it is zero, we jump to the location pointed to by immediate evalue else next instruction is fetched for the execution. The contents of register Rz are pushed on TOS for checking and immediate value is loaded into T. If the TOS is zero, T is loaded into JPC and becomes the address of the next PC. | nop opd  rdrf  ldrzt  ldt  ldctrlopdt  nop  jbr  nop  nop  pop nxt | Rz🡪 T  T🡪 A  Ctrlopd 🡪 T  T 🡪 JPC (if A= 0) |
| **24** | **jopsys\_sz**  **CRCF**  This instruction performs a conditional jump. This instruction always follows the immediate subtract instruction which affect the zero flag. The if zero flag is set, we jump to the location pointed to by immediate value else next instruction is fetched for the execution. The immediate value is loaded into T. If the zero flag is set, T becomes the next PC. | nop opd  ldctrlopdt  nop  jbr  nop  nop nxt | Ctrlopd 🡪 T  T 🡪 JPC ( if zf = 1)  Remove the |
| **25** | **jopsys\_subimm**  **CRCF**  It performs AND operation between the immediate value and content of register Rx and stored the result into the register Rz. The content register Rx and operand values are pushe onto stack and AND operand is performed. The is result available on TOS The result is no written back to memory. It affects the Zero flag. | nop opd  rdrf  nop  ldrxt  ldt  ldctrlopdt  ldt  sub  nop  stzf  nop  pop nxt | Rx 🡪 T  T🡪 A(=Rx)  ctrlopd🡪 T  T🡪 A (=ctrlopd), B= Rx  A 🡪 A-B (= Rx+ctrlopd) (set ZF if A=0) |
| **26** | **jopsys\_chkend**  **CRCF**  This instruction find the maximum of Rx and Rz{3..0} nibbles and results is stored in Rz. The contents of Rz and Rx are fed to special hardware unit which finds the maximum nibble which is loaded into T. It is then pushed onto stack from where it is written to register file. | rdrf  nop  ldmaxt  ldt  nop  wrrf  nop  pop nxt | Maxvlue 🡪 T  T🡪 A  A 🡪 RF[z] |
| **27** | **jopsys\_switchs**  **CRCF**  This instruction is a complex and results in a un-conditional jump together with couple of memory read operation. The contents of register Rx are pushed on stack and the memory location pointed to by its contents (parent node) is read into register T which contains the number of child node we want to switch. This is added to parent node incremented by one (child nodes are stored next to parent codes). This value is the address where the pointer to the selected child node is stored. The address is read into T and unconditional jump results in execution from the address provided by T. | rdrf  ldrxt  ldt  nop  nop  lddmt  nop  ldt  add  nop  ldonet  ldt  nop  add  nop  nop  lddmt  ldt  wrrf  jbr  nop  pop  pop nxt | Rx🡪 T  T🡪 A(rx=address of parent)  dm[A] 🡪 T (A=Rx)  T🡪 A (A= switch case child number), B=parent  A+B 🡪 A (parent + child) pointer to child pointer-1  A🡪 A+1 A(=corrected pointer child pointer)  Dm[A] 🡪 T child node address  T 🡪 A  A 🡪 RF[z]  T 🡪 JPC (child node address)  Remove address  Remove correct pointer |
| **28** | **jopsys\_clfz**  **CRCF**  This instruction clears the zero flag. | clzf  nop nxt | 0 🡪 ZFreg |
| **29** | **jopsys\_seot**  **CRCF**  This instruction sets the eot. | seot  nop nxt |  |
| **30** | **jopsys\_ceot** | ceot  nop nxt |  |
| **31** | **jopsys\_ler** | ldert  ldt  wrrf  nop  pop nxt |  |
| **32** | **jopsys\_cer** | cer  nop nxt |  |
| **33** | **jopsys\_ssvop** | ssvop  nop nxt |  |
| **34** | **jopsys\_ssop** | sop  nop nxt |  |
| **35** | **jopsys\_lsip** | ldsipt  ldt  wrrf  nop  pop nxt |  |
| **36** | **jopsys\_cinit** | nop opd  nop  nop nxt |  |
| **37** | **jopsys\_esl** | nop  nop nxt |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

After the JOP bootup, the main method is invoked. The mere task of the main is to initialize the control memory and hands over the control to CRCF. The main is never invoked after that as the galsjop+ introduces an efficient mechanism of invoking dpcf. Previously it was done through the main method. Where a data computation requires returning to main which in turn invokes the required method.

During the start up, the pointer to main method data structures is stored in a register which is used to calculate the address of the data structure to be invoked.